



Gymnázium F. X. Šaldy

LIBEREC 11, PARTYZÁNSKÁ 530, PŘÍSPĚVKOVÁ ORGANIZACE

Kristian Koutek

Kontejnery

Maturitní práce

Vedoucí práce: Mgr. Vítězslav Pěnička

Liberec 2025

Zadání maturitní práce

Školní rok: 2024–2025 **Třída:** 4. B
Jméno a příjmení: Kristian Koutek
Maturitní předmět: Aplikovaná matematika
Téma maturitní práce: Kontejnery
Termín odevzdání: do 17. 3. 2025 do 14.00 v místnosti C7
Délka obhajoby: 15 min
Vedoucí práce: Mgr. Vítězslav Pěnička
Požadovaný rozsah práce: minimálně 20 normostran

Pokyny k obsahu práce:

V teoretické části maturant definuje základní pojmy z oblasti kontejnerizace, popíše rozdíly mezi Podmanem a Dockerem. V praktické části popíše instalaci OS, Podmanu a Dockeru. Maturant popíše umělou zátěž v kontejnerizovaném prostředí a vyhodnotí toto měření. Autor vytvoří výukovou prezentaci a pracovní listy na téma „Kontejnery v IT“ pro žáky semináře z informatiky.

Způsob zpracování: odborný text, prezentace, pracovní listy

Určení části tématu: žák zpracuje maturitní práci individuálně.

Počet vyhotovení práce: Dva tištěné výtisky (v kroužkové vazbě nebo v termovazbě). Elektronická verze práce odevzdaná prostřednictvím Učebny Google nebo na CD/DVD.

Na CD/DVD bude uložen text práce ve formátu PDF i v původním formátu (např. docx) a připojeny příp. další soubory stanovené v tomto zadání (např. zdrojové kódy, fotodokumentace v původním rozlišení, postery). — Práce, jejichž součástí nejsou rozsáhlé přílohy, lze alternativně odevzdat prostřednictvím Učebny Google v kurzu pro maturitní ročníky.

Kritéria hodnocení:

1. splnění požadavků na práci stanovených v tomto zadání,
2. správné užití terminologie, kritické užití literatury a dalších zdrojů informací,
3. jednotná a jednoznačná identifikace literatury, pramenů a dalších zdrojů, důsledné odlišení vlastních autorských myšlenek od citací a parafrází myšlenek jiných autorů,

4. přínos práce (tvůrčí přístup, přidaná hodnota, využití pro praxi),
5. jazyková úroveň (pravopis, odborný styl),
6. struktura práce (logické uspořádání kapitol, přehlednost, textová návaznost),
7. dodržení typografických pravidel,
8. obhajoba práce (přípravená prezentace, projev, reakce na otázky komise).

V případě, že práce

- a) podstatně nesplňuje požadavky uvedené v tomto zadání,
- b) vykazuje závažné prohřešky proti citační etice (citace nejsou v textu práce označeny a identifikovány; parafráze cizích myšlenek jsou bez označení zdroje vydávány za myšlenky vlastní),

bude hodnocena celkovou známkou nedostatečně (bez ohledu na hodnocení ostatních dílčích kritérií).

V Liberci dne 16. prosince 2024



ředitel školy



vedoucí práce



žák/yně

Prohlášení autora

Byl jsem seznámen s tím, že na mou maturitní práci se vztahuje zákon č. 121/2000 Sb., o právu autorském, právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména § 60 – školní dílo. Beru na vědomí, že Gymnázium F. X. Šaldy nezasahuje do mých autorských práv užitím mé maturitní práce pro potřeby školy. Užiji-li maturitní práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Gymnázium F. X. Šaldy. Maturitní práci jsem vypracoval samostatně s použitím uvedené literatury a dalších zdrojů a na základě konzultací s vedoucím práce, popř. dalšími konzultanty, kteří jsou v práci uvedeni.

V Liberci dne 6. 3. 2025

Anotace

Tato práce porovnává dvě kontejnerové technologie, Docker a Podman, z hlediska využití prostředků a provozní stability. Úvodní část pojednává o rychlém tempu vývoje technologií a jeho vlivu na průmyslové standardy s důrazem na roli kontejnerizace v moderním vývoji softwaru. Následující část přináší základní poznatky o kontejnerech a také rozlišení technologií Docker a Podman včetně jejich architektury a výhod.

Poté se text zaměřuje na nastavení testovacího prostředí, kde popisuje proces instalace operačního systému Debian GNU/Linux a následnou instalaci kontejnerizačních enginů. Je důležité poznamenat, že při přípravě tohoto testovacího nastavení pro obě technologie je třeba mít na paměti ekvivalenci. Je zde také popsáno nasazení monitorovacích nástrojů, jako jsou Grafana a Prometheus pro sběr a vizualizaci dat.

Dále následuje vyhodnocení naměřených dat, které porovnává využití prostředků Docker a Podman. Grafy jsou prezentovány spolu s komentáři, které poskytují přehled o chování jednotlivých technologií během testovací fáze. Nakonec jsou uvedeny diskuse o zjištěních, které shrnují klíčové rozdíly pozorované během experimentu spolu s případnými zajímavými jevy, na které se narazilo. Součástí práce je i výuková prezentace a pracovní list pro výuku kontejnerizace v hodinách ICT.

Abstract

This work compares two containerization technologies, Docker and Podman, in terms of resource utilization and operational stability. The introductory section discusses the fast pace of technology development and its effect on industrial standards with emphasis on the role of containerization in modern software development. The following part gives a basic knowledge about containers as well as differentiating between Docker and Podman including their architectures and benefits.

The text then focuses on setting up a test environment, where it describes installation process for Debian GNU/Linux operating system and subsequent installation of containerization engines. It is important to note that equivalence should be kept in mind when preparing this test setup for both technologies. Deployment of monitoring tools such as Grafana and Prometheus for data collection & visualization is also covered here.

Measured data evaluation which compares resource usage of Docker with Podman follows next. Graphs are presented together with comments that give insight on behaviour of each technology during testing phase. Lastly findings discussions are presented which summarize key differences observed during experiment along with any interesting phenomenon encountered. The work also includes an educational presentation and a worksheet for teaching containerization in ICT classes.

Obsah

Anotace.....	7
Abstract.....	7
Obsah.....	9
Úvod.....	10
1. O kontejnerizaci.....	11
1.1 O rozdílu mezi Podmanem a Dockerem.....	13
2. Nasazení testovacího prostředí.....	19
2.2 Konfigurace sítě pro sestavu.....	21
2.3 Instalace kontejnerizačního enginu (společná část).....	23
2.3.1 Instalace Podmanu.....	24
2.3.2 Instalace Dockeru.....	25
2.4 Instalace a nastavení monitoringu (společná část).....	26
2.5 Spuštění umělé zátěže v kontejnerizovaném prostředí (společná část).....	28
3. Vyhodnocení měření.....	29
3. 1. Vyhodnocení měření na technologii Docker.....	30
3. 2. Vyhodnocení měření na technologii Podman.....	34
4. Zpracování výukové prezentace.....	37
5. Zpracování pracovních listů pro výuku kontejnerizace.....	38
Závěr.....	39
Conclusion.....	39
Zdroje.....	41
Obrazové zdroje.....	41

Úvod

Zastávám názor, že vzhledem k aktuálnímu rychlému vývoji technologií se v dnešní době na pozici takzvaného průmyslového standardu udrží málokteré řešení po delší dobu. Neustálý vývoj softwaru a hardwaru totiž přináší inovace, jež často způsobí, že se konkurenční řešení stane signifikantně výhodnějším než to aktuálně používané. Tato dynamika je patrná nejen ve světě serverových technologií, ale i v oblastech jako je cloud computing nebo virtualizace, kde se každý rok objevují nové přístupy a nástroje, jež redefinují standardy.

Jednou z novějších technologií, která zásadním způsobem urychluje vývoj softwarových aplikací, je kontejnerizace (Šípek 2018). Tento přístup umožňuje vývojářům zabalit aplikaci spolu se všemi jejími závislostmi do samostatného, přenosného balíku. Díky tomu se výrazně zjednodušuje nejen proces vývoje a testování, ale také samotné nasazení do produkčního prostředí. Kontejnerizace navíc umožňuje škálování výpočetních zdrojů jak vertikálně, tedy zvýšením výkonu jednotlivých serverů, tak horizontálně, kdy se zátěž rozprostírá mezi více zařízeními. Tato flexibilita přináší nemalé výhody zejména pro projekty s proměnlivým nebo nárazovým zatížením, jako jsou například streamovací služby či aplikace s výraznou sezónní dynamikou.

Jakkoli je téma škálovatelnosti samo o sobě fascinující, já bych raději věnoval pozornost problematice zkoumání rozdílnosti charakteristiky využití zdrojů u dvou konkurenčních kontejnerizačních technologií. Abych mohl tento výzkum prakticky provést, je nezbytné obě technologie – Docker a Podman – nasadit do prostředí, kde budou vystaveny ekvivalentní zátěži. Tento přístup mi umožní získat přesná data potřebná ke komparaci nejen samotného výkonu a efektivity, ale také ke srovnání metodických postupů při jejich implementaci a následné správě.

Považuji za klíčové, aby byl metodický postup sám o sobě součástí analýzy, neboť umožňuje replikaci výsledků a potvrzení či vyvrácení mých závěrů. Během samotného nasazení se totiž často odhalují další, méně zjevné rozdíly, které mohou mít zásadní dopad na provoz a udržitelnost celého systému.

1. O kontejnerizaci

Kontejnerizace je způsob izolace běhu programu či skupiny programů, který má na rozdíl od jiných metod izolace, jako je hypervisor, jen výrazně nižší nároky na systémové zdroje (Arora 2023). Abychom si kontejnerizaci lépe představili, můžeme použít přirovnání o plachetnicích. Když izolujeme běh aplikace pomocí hypervizoru, je to podobné, jako bychom chtěli po řece přepravit 80 různých chemikálií, přičemž nechceme, aby se vzájemně promíchaly. Proto bychom v přístavu připravili 80 malých plachetnic, na každou naložili jednu chemikálii – výsledkem je flotila osmdesáti plachetnic, přičemž každá má svůj vlastní motor, kapitána a posádku. Kolem se šíří smog, komunikace mezi loděmi se komplikuje a prostor na řece je značně omezený. Pokud bychom však všechny chemikálie naložili do standardizovaných lodních kontejnerů a ty pak naložili na jednu moderní velkou plachetnici s jedním silným motorem, jedním lodníkem a jedním kapitánem, zredukovali bychom jak náklady, tak využití zdrojů – to je podstata kontejnerizace. Ano, v tomto příkladu jsou použity speciální motorové bezplachtové plachetnice.

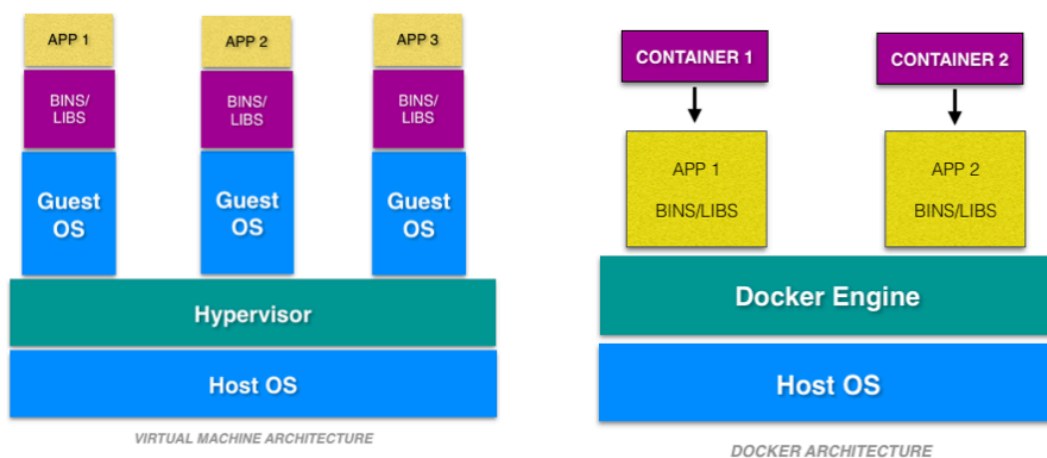
Toto přirovnání nejenže krásně ilustruje, jak kontejnerizace šetří zdroje, ale také zdůrazňuje, že kontejner sdílí základní funkce systému s ostatními izolovanými aplikacemi. Není nutné, aby každá aplikace měla vlastní operační systém, síťové rozhraní nebo dokonce zálohovací program, což vede ke snížení režijních nákladů a ke zvýšení efektivity. Zároveň kontejner zajišťuje stabilní a neměnné podmínky pro běh aplikace, což umožňuje, aby vývojář A, jenž objeví chybu ve svém kódu, mohl poslat soubor ve formě dockerfile vývojáři B. Ten aby pak díky shodným podmínkám bez problémů replikoval chybu a následně zaslal opravenou verzi (Čech 2021).

V současném prostředí, kde se vyvíjí stále složitější aplikace, je třeba zajistit nejen správu jednotlivých kontejnerů, ale také jejich orchestraci – tedy koordinaci a řízení celé infrastruktury, která může zahrnovat desítky či stovky kontejnerů. Zde přicházejí do hry různé nástroje a platformy pro orchestraci. Například Docker Compose je nástroj, který umožňuje definovat a spouštět více kontejnerové aplikace pomocí jednoduchého YAML souboru. Umožňuje tak snadné nastavení závislostí mezi kontejnery, definici sítí a objemů dat, což značně zjednodušuje vývoj a testování komplexních aplikací.

Pro produkční nasazení a škálování jsou však často voleny robustnější platformy, jako je Kubernetes. Kubernetes poskytuje rozsáhlé možnosti pro automatizaci nasazení, škálování a

správu kontejnerizovaných aplikací. V této souvislosti se objevují i lehčí varianty, například k3s, což je certifikovaná a kompaktní distribuce Kubernetes určená pro zdrojově omezené prostředí, jako jsou IoT zařízení či okrajové servery. Díky své modulární architektuře a bohatému ekosystému umožňují tato řešení efektivně využívat dostupné zdroje a současně zaručují vysokou úroveň spolehlivosti a dostupnosti.

Dalšími nástroji orchestrace mohou být Docker Swarm či nástroje jako Nomad, které nabízejí jednodušší alternativy pro menší infrastruktury, kde není třeba rozsáhlé funkcionality Kubernetes. Každé z těchto řešení má své specifické výhody a nevýhody, a proto je klíčové zvolit takové, které nejlépe odpovídá potřebám konkrétního projektu. Celkově tedy kontejnerizace nejenže optimalizuje využití systémových zdrojů, ale spolu s moderními orchestračními nástroji umožňuje i efektivní a pružné nasazení, správu a škálování aplikací v dynamicky se měnícím IT prostředí.



Obrázek 1.: Hypervizor vs konteiner (ARORA 2023)

1.1 O rozdílu mezi Podmanem a Dockerem

Zřejmě nejméně podstatným, avšak zajímavým rozdílem mezi těmito dvěma technologiemi je skutečnost, že Podman je vyvíjen společností Red Hat jako náhrada Dockeru určená primárně pro provoz některých jejích proprietárních softwarových řešení, například pro řízení domácích spotřebičů, jako jsou pračky či ledničky. Naopak technologie Docker, vyvíjená společností Docker.inc, vychází z modelu komunitního vývoje, což se promítá do jejího způsobu aktualizací, zpětné kompatibility a celkové filozofie řízení. Oba tyto open source nástroje lze bez váhání označit za „švýcarské nože“ v arzenálu každého specialisty na dev-ops, neboť nabízejí řadu funkcí, které výrazně usnadňují práci se správou a nasazováním kontejnerových aplikací.

Mezi klíčové rozdíly patří zejména způsob správy procesů. Docker totiž vyžaduje pro své fungování tzv. demony, tedy centralizované procesy běžící s rozšířenými oprávněními, což může představovat potenciální bezpečnostní riziko. Naproti tomu Podman umožňuje provozovat kontejnery přímo pod uživatelskými právy, neboť funguje v režimu rootless od samého počátku. Tato vlastnost zásadně zvyšuje bezpečnost celého systému, protože omezuje možnosti útoku prostřednictvím privilegovaných procesů. Nicméně absence démonů v Podmanu přináší nutnost spoléhat se na jiné mechanismy volání procesů, přičemž zde hraje významnou roli systemd, který zajišťuje správu a koordinaci procesů efektivním způsobem.

Dalším aspektem je využití privilegovaných operací. Docker totiž přiřazuje určité privilegia, která mohou být potenciální branou pro útoky, a přestože některé typy interakcí mezi kontejnery v prostředí Dockeru fungují díky garantovaným oprávněním stabilně a spolehlivě, celková bezpečnost systému může být tímto přístupem oslabená. Zároveň však prostředí Dockeru těží z rozsáhlé podpory orchestračních nástrojů, což usnadňuje automatizaci a škálování rozsáhlejších systémů, i když toto řešení bývá někdy pro uživatele nepřívětivé v případě přechodu na Podman, kde integrace těchto nástrojů není tak intuitivní.

Co se týče samonosnosti, Docker působí jako samostatné řešení, které v mnoha případech dovoluje rychlé nasazení bez nutnosti složitého nastavování dalších komponent. Na druhé straně Podman, ačkoli pracuje s velmi podobnými příkazy a ovládacími prvky, se vyznačuje modulárním přístupem. To znamená, že pro sestavení kontejneru nestačí pouze jeho nativní technologie – je zde třeba využít doplňkový software, například Buildah, který rozšiřuje možnosti konfigurace a umožňuje preciznější kontrolu nad výsledným obrazem

kontejneru. Takový modulární přístup poskytuje větší flexibilitu, avšak zároveň klade vyšší nároky na znalosti a zkušenosti správce systému, což je třeba pečlivě zvážit při výběru technologie pro konkrétní nasazení. (Gamela 2023)

1.2 Míra nasazení technologií v produkci

Docker je v oblasti kontejnerizace již dlouho dominantním hráčem, přičemž svou popularitu získal zejména díky přívětivému uživatelskému rozhraní, rozsáhlé dokumentaci a silné podpoře komunity. Tento přístup ocení zejména začátečníci a malé týmy, které hledají rychlé a efektivní řešení pro nasazení aplikací bez zbytečné komplikovanosti.

Na druhou stranu Podman, vyvíjený hlavně společností Red Hat, představuje novější a modernější přístup. Jeho hlavní výhodou je absence nutnosti běžícího demona, což zvyšuje bezpečnost a zjednodušuje údržbu v prostředích, kde se žádná dlouhodobě běžící služba nechce stát potenciálním cílem útoku. Díky integraci s nástroji jako je systemd, umožňuje Podman efektivní správu kontejnerů přímo v rámci stávajícího systémového prostředí, což je obzvláště užitečné při produkčních nasazeních. Vzhledem k rozsáhlejším možnostem pokročilé orchestrace vyhrává souboj o současnou nasazenost Docker, což se však může rychle změnit vzhledem k téměř raketovému vývoji Podman ekosystému stran společnosti Red Hat. (Dawson 2022)

V rámci této práce jsem také provedl dotazníkový průzkum mezi třiceti mně dostupnými odborníky z různých oblastí IT. Průzkum měl za cíl zjistit jejich zkušenosti a preference při používání obou těchto technologií. Dotazník se zaměřoval například na jednoduchost nasazení, bezpečnostní aspekty, možnosti správy a škálovatelnost řešení. Výsledky ukázaly, že zatímco Docker je oblíbený zejména pro svou uživatelskou přívětivost a rozsáhlou komunitní podporu, Podman získává na atraktivitě zejména v prostředích, kde hraje roli bezpečnost a integrace s moderními systémy správy. Tento průzkum tak poskytl cenné poznatky o aktuálních trendech v oblasti kontejnerizace a podtrhuje, že volba technologie by měla vždy reflektovat specifické potřeby daného projektu.

Níže uvádím optimalizovaný přehled výsledků dotazníkového šetření, kterého se zúčastnilo celkem 30 respondentů. Pro přehlednost jsem jednotlivé otázky rozdělil do samostatných tabulek:

1. Jak často aktualizujete technologie ve svém pracovním prostředí?

Frekvence aktualizace	Počet respondentů
Častěji než jednou ročně	5
Jednou za dva roky	15
Jednou za tři až pět let	7
Neaktualizuji	3

2. Jaké faktory berete v úvahu při výběru technologií pro konkrétní projekty?

(Respondenti mohli vybrat více možností)

Faktor	Počet respondentů
Kritičnost projektu	24
Potřeba datové bezpečnosti	18
Osobní preference	15

Průmyslové standardy / ISO	12
----------------------------	----

3. Používáte kontejnerizační technologii?

Odpověď	Počet respondentů
Ano	30
Ne	0

4. Když používáte kontejnerizační technologii, jaké nejčastěji preferujete?

Technologie	Počet respondentů
Docker	10
Podman	12
rkt	1
Jiné (LXC, OpenVZ, ...)	7

5. Jaké jsou vaše zkušenosti s nasazováním a správou Dockeru a Podmanu v produkčním prostředí?

Zkušenost	Počet respondentů
Žádné	3
Velmi pozitivní	10
Pozitivní	8
Negativní	5
Velmi negativní	4

Tyto tabulky poskytují komplexní přehled o preferencích a zkušenostech respondentů. Výsledky jasně ukazují, že většina účastníků pravidelně aktualizuje technologie (převážně jednou za dva roky) a klade největší důraz na kritičnost projektu a bezpečnost dat. Také je patrné, že v oblasti kontejnerizačních technologií dominují Docker a Podman, přičemž zkušenosti s jejich nasazováním jsou převážně pozitivní.

2. Nasazení testovacího prostředí

V následující kapitole popisují podrobný postup nasazení technologie a způsob sběru dat, který byl klíčovým prvkem mého experimentu. Celá aktivita byla rozdělena na dvě sobě jdoucí testovací sekvence, přičemž se vždy testovaly dvě různé aplikace. První částí experimentu bylo nasazení aplikací v prostředí Docker, následované jejich nasazením v prostředí Podman. Tento přístup zajistil přesnou ekvivalenci testovací sestavy pro obě technologie, neboť konfigurace testovacích aplikací byla v obou případech záměrně identická.

Nasazení probíhalo na lokálním stroji, což představovalo optimální volbu z hlediska stability a konzistence dostupných zdrojů. Při použití vzdálených serverů, jako jsou VPS, by hrozilo riziko kolísání výpočetní kapacity v důsledku sdílení zdrojů mezi více uživateli. Takové proměnlivé podmínky by mohly výrazně narušit přesnost naměřených hodnot a znemožnit validní interpretaci výsledků, a proto jsem se rozhodl pro lokální prostředí, které minimalizuje vnější vlivy a umožňuje opakování experimentu za stejných podmínek.

Pro přehlednost je celá kapitola členěna do společných a oddělených částí. Společná část popisuje jednotnou metodiku a konfiguraci testovacího prostředí, zatímco oddělená část se věnuje specifickým výsledkům a rozdílům pozorovaným při nasazení obou technologií.

2.1 Instalace OS (společná část)

Rozhodl jsem se testování provést na operačním systému Debian GNU/Linux, protože se jedná o velmi stabilní a uživatelsky přívětivou distribuci s dalekosáhlou historií a aktivní komunitou vývojářů. Debian je svobodný operační systém. Operační systém je sada základních programů a pomůcek, které k provozu potřebuje každý počítač. Základem operačního systému Debian je jádro, které zajišťuje všechny jeho základní služby a umožňuje bezproblémový běh ostatních aplikací. Díky robustní architektuře a dlouhodobé podpoře komunity se Debian stal ideální volbou pro testovací prostředí, kde se klade důraz na stabilitu a spolehlivost. (Debian 2023)

Vzhledem k malému rozsahu testu, který byl realizován na jediném fyzickém stroji, jsem se rozhodl pro instalaci systému z fyzického instalačního média místo klasické síťové instalace. Tento přístup nejenže zjednodušil celý proces, ale také umožnil přesnější kontrolu nad instalací, protože všechny potřebné komponenty byly získány z ověřených zdrojů. S ohledem na vysokou rychlost internetu, kterou testovací sestava disponuje, jsem využil takzvaný minimální instalační obraz. Tento obraz, stažený z oficiálních stránek projektu, funguje na principu stažení většiny knihoven a komponent až během samotné instalace, čímž se výrazně snižují nároky na velikost paměti instalačního média.

Další části systému jsem následně instaloval z lokálního zrcadla provozovaného studentským spolkem SiliconHill při Pražském ČVUT, což bylo zvoleno výhradně z důvodu rychlosti a spolehlivosti stahovaných balíků. Vzhledem k tomu, že se jedná o porovnání technologií v primárně produkčním, nikoliv uživatelském, prostředí, jsem se rozhodl instalovat operační systém bez zbytečných nadbytečností, tedy bez grafického rozhraní, aby nedocházelo k nežádoucímu zatížení systémových zdrojů.

Přístup do testovací soustavy byl následně zajištěn pomocí SSH připojení, které bylo zabezpečeno dvojicí SSH klíčů. V rámci bezpečnostních opatření bylo proto nutné nainstalovat a nakonfigurovat OpenSSH Server. Tento nástroj byl dlouhou dobu průmyslovým standardem v oblasti vzdáleného přístupu k výpočetnímu výkonu, přičemž jeho robustnost a osvědčenost z něj činily ideální volbu pro naše účely. Modernější a podstatně bezpečnější technologie, které dnes postupně nahrazují či doplňují OpenSSH, by v kontextu tohoto testu představovaly jen nadbytečnou komplikaci, jež by neměla vliv samotné srovnání technologických řešení. (Blake 2022)

2.2 Konfigurace sítě pro sestavu

Aby bylo možné připojit vzdálenou umělou zátěž k serveru, bylo nejprve nutné nalézt vhodné řešení, které umožní přístup k serveru i bez veřejné IP adresy. Na počátku jsem zvažoval několik variant, mezi nimiž vynikal tzv. cloudflare tunel. Tato metoda totiž poskytuje možnost trvalého zpřístupnění infrastruktury nacházející se v privátní síti přímo do internetu, což by bylo ideální pro produkční scénáře. Nicméně, implementace tohoto řešení by si vyžádala složitější konfiguraci, zahrnující dodatečné bezpečnostní a síťové vrstvy, což by pro účely této práce znamenalo zbytečné komplikace.

S ohledem na uvedené skutečnosti jsem se rozhodl pro alternativní řešení, které kombinuje jednoduchost nasazení s vysokou úrovní bezpečnosti a provozní stability. Pro připojení vzdálené zátěže jsem zvolil VPN tunel využívající technologii WireGuard. Tento moderní protokol je známý svou nízkou latencí, vysokou rychlostí a efektivním šifrováním dat, což představuje klíčový předpoklad pro bezpečné a spolehlivé spojení. Tunel jsem terminoval přímo na routeru umístěném před samotným serverem, čímž jsem se efektivně zbavil potřeby řešit konfiguraci sítě uvnitř serveru a k privátní IP adrese, kterou jsem serveru ručně v rámci VPN sítě přidělil, jsem se po zbytek experimentu mohl chovat, jakoby se jednalo o veřejnou IP adresu.

Pro správu a řízení tohoto tunelu jsem využil službu Tailscale, která významně zjednodušuje konfiguraci a správu přístupů v rámci VPN sítí. Díky svému přehlednému grafickému rozhraní umožňuje Tailscale nejen rychlé nasazení, ale také flexibilní monitorování a správu jednotlivých uzlů v síti, což je při implementaci distribuované umělé zátěže klíčové. Uživatelská přívětivost této služby přispěla k eliminaci potenciálních chyb, které by mohly ohrozit stabilitu celého systému v průběhu měření.

Během praktických testů se osvědčila efektivita zvoleného řešení, neboť VPN tunel postavený na WireGuardu poskytoval konzistentní výkon, a plynulost služby na straně klienta byla ovlivněna srovnatelně se situací, kdy je před servery služby umístěn loadbalancer, který by tam v případě produkčního nasazení dozajista byl. Ve srovnání s alternativami, jako je cloudflare tunel, se tato varianta ukázala jako mnohem jednodušší na implementaci a správu. Díky kombinaci těchto technologií se podařilo dosáhnout spolehlivého přístupu k serveru, který byl zároveň flexibilní a připraven reagovat na případné změny v provozní infrastruktuře. Tento přístup tedy představuje ideální kompromis mezi jednoduchostí

konfigurace, provozní efektivitou a robustností celého systému, což výrazně přispělo k úspěchu celé práce. Tento postup byl zvolen proto, že se jedná o experiment v uzavřené síti, a pro produkční účely jiné než provoz tohoto typu služby pro použití vlastní nebo použití omezenou skupinou uživatelů by byl zcela nevhodný.

2.3 Instalace kontejnerizačního enginu (společná část)

Po instalaci operačního systému je na testovacím serveru prvotně nutné nainstalovat engine zvolené kontejnerizační technologie. Naštěstí pro tento projekt je Podman dostupný přímo v apt depozitáři, což umožňuje jeho instalaci jednoduchým příkazem. Docker sice v základním apt depozitáři není, ovšem své apt depozitáře má a není problém je do systému několik příkazy přidat. Tento krok výrazně urychluje počáteční nastavení serveru a eliminuje potřebu složitých manuálních zásahů, které by jinak mohly prodloužit dobu potřebnou k nasazení testovacího prostředí.

Abychom si následně usnadnili práci s kontejnerovou infrastrukturou, rozhodli jsme se spustit v kontejneru i grafické uživatelské rozhraní. Toto GUI nabízí přehledné zobrazení všech běžících kontejnerů, virtuálních disků i síťových rozhraní, což značně zjednodušuje další operace a monitorování celého systému. Do té doby totiž vše ovládáme prostřednictvím terminálu, který sice poskytuje vysokou míru kontroly a rychlost, ale pro některé uživatele může být z hlediska orientace poněkud matoucí.

Pro tento účel jsme zvolili nástroj Portainer. Jeho intuitivní a přehledné grafické rozhraní umožňuje snadnou správu kontejnerů, což výrazně zjednodušuje běžnou administraci. Přitom je však třeba mít na paměti, že je nutné pečlivě sledovat, které kontejnery byly spuštěny pomocí Portaineru, a které prostřednictvím terminálu. Kontejnery inicializované pomocí Portaineru lze totiž následně upravovat i z terminálu, avšak opačný postup není možný. Tento aspekt je zásadní, aby nedošlo k nechtěným kolizím při správě konfigurací a aktualizaci jednotlivých služeb.

Je důležité rovněž zdůraznit, že se jedná o komunitní edici Portaineru, která je pro naše účely zcela dostačující. Komerční verze, i když nabízí řadu rozšířených funkcí, je totiž zaměřena primárně na správu kontejnerů v prostředí rozsáhlých serverových clusterů a kubernetes či jiných rozsáhlých orchestračních řešení, a takové řešení by bylo v tomto kontextu nevhodné. Zvolení komunitní edice tedy představuje ideální kompromis mezi jednoduchostí nasazení, efektivitou správy a nutnou funkčností, kterou základní provoz kontejnerů vyžaduje. Celkově tak tento přístup nejenže zefektivňuje správu testovacího serveru, ale také umožňuje uživatelům pracovat v prostředí, které je zároveň výkonné i uživatelsky přívětivé.

2.3.1 Instalace Podmanu

Podman nainstalujeme jednoduchým příkazem:

```
Sudo apt install podman
```

Nyní, když je Podman nainstalovaný, posuneme se k instalaci portainer. Přidáme do souboru zdrojů již sestavených image cestu k Docker Hub, ze kterého získáme docker image, na podmanu použijeme docker image. Tedy si otevřeme tento soubor, což musíme to udělat jako superuser, aby nám bylo umožněno upravit tento soubor.

```
sudo nano /etc/containers/registries.conf
```

Na jeho konec přidáme následující řádky, které nám umožní stahovat image z docker hub.

```
[registries.search]
registries = ['docker.io']
```

Poté už jen stačí spustit instalační příkaz, který je nutné oproti dokumentaci upravit, protože docker v základu počítá s rootfull možnostmi, a my se snažíme službu spustit v podmanu tedy rootles prostředí - pro aktuální informaci vizte oficiální dokumentaci podmanu od společnosti redhat. Kontejner spouštíme v root modu jen proto, že on sám bude spouštět další kontejnery, a to by bylo bez root privilegií složité, navíc jsou bezpečnostní rizika tohoto řešení a dalších orchestračních řešení bez rootu přibližně stejná a v tomto případě je můžeme považovat za nepodstatná.

```
sudo podman run -d -p 9443:9443 --privileged -v /run/podman/podman.sock:/var/run/docker.sock:Z portainer/portainer-ce
```

Zde jsme se setkali s následující chybovou hláškou, která informuje o absenci platných certifikátů:

```
Error: initializing source docker://portainer/portainer-ce:latest: pinging container registry registry-1.docker.io: Get "https://registry-1.docker.io/v2/": x509: certificate signed by unknown authority
```

Vyřešili jsme ji pomocí triviální reinstalace certifikátů:

```
sudo apt update
sudo apt install --reinstall ca-certificates
```

2.3.2 Instalace Dockeru

Docker nainstalujeme jednoduchým příkazem:

```
Sudo apt install docker.
```

Nyní, když je Docker nainstalovaný, posuneme se k instalaci portainer instalačním příkazem.

```
sudo docker run -d -p 9443:9443 portainer/portainer-ce
```

2.4 Instalace a nastavení monitoringu (společná část)

Pro lepší přehled o datech jsem zvolil nástroj zvaný grafana, který sice není nejsilnější v oblasti raw dat, je ovšem skvělý v oblasti grafů. Grafy jsou pro zamýšlený účel zřejmě o něco vhodnější než tabulky s daty. Za podstatné metriky považuji zejména utilizaci RAM a procesorového času. Podstatou měření je zjištění rozdílů mezi Podmanem a Dockerem během transcodingu/downscalingu videa. Tento typ procesu jsem zvolil ze dvou poměrně prozaických důvodů - jedná se o z hw hlediska náročný proces a využijí ho aplikace streamující videoobsah k uživateli, což jsou vhodné kandidáty na kontejnerizaci.

Grafanu instalujeme pomocí následujících tří příkazů, musíme dbát zvýšené pozornosti, abychom nainstalovali a spustili komunitní verzi tohoto sw:

```
sudo apt-get install -y adduser libfontconfig1 musl
wget https://dl.grafana.com/oss/release/grafana_10.4.2_amd64.deb
sudo dpkg -i grafana_10.4.2_amd64.deb
```

Následující konfigurací zajistíme start aplikace při nabootování serveru.

```
sudo systemctl enable grafana-server.service
```

(Grafana 2023)

Vzhledem ke skutečnosti, že grafana je nástroj primárně pro upozornění a zobrazení údajů monitoringu, musíme spustit ještě sw, který nám posbírání data na vstupu - Prometheus. Instalace Promethea je složitá, tedy využijeme výhod dockeru, konkrétně nástroje docker compose a nasadíme aplikaci dle předem připravených obrazů a konfigurací.

Dockercompose file:

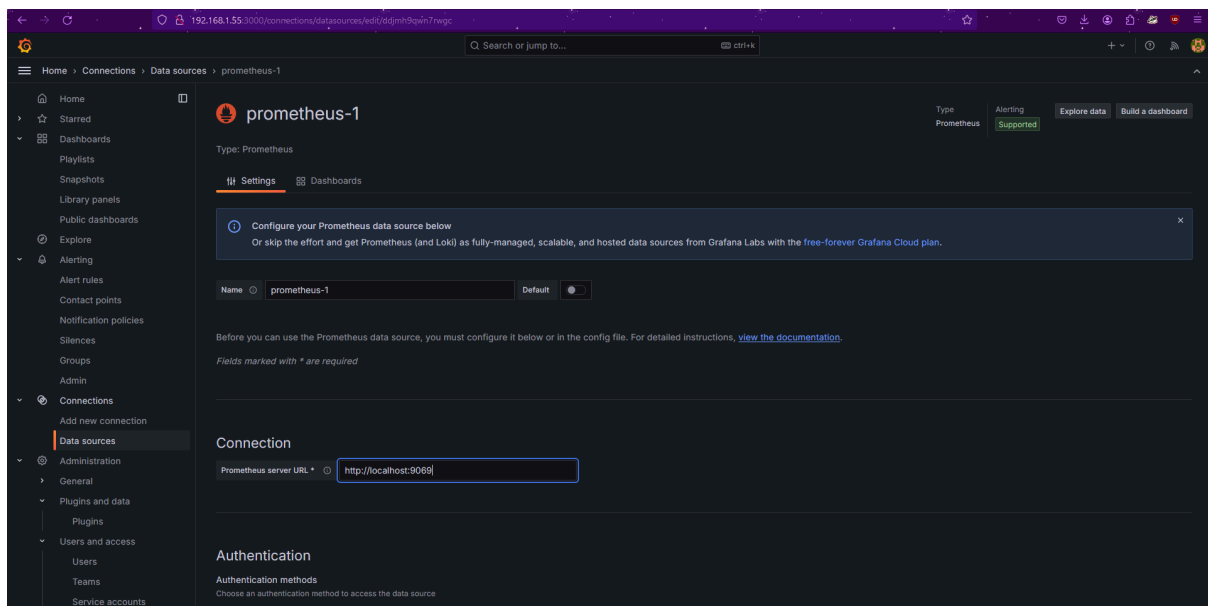
```
---
volumes:
  prometheus-data:
    driver: local
services:
  prometheus:
    image: docker.io/prom/prometheus:v2.51.2
    container_name: prometheus
    ports:
```

```
- 9069:9090

# command: "--config.file=/etc/prometheus/prometheus.yaml"
volumes:
- ./config/prometheus.yaml:/etc/prometheus/prometheus.yaml
- prometheus-data:/prometheus

restart: unless-stopped
```

Prometheus do Grafany připojíme přes rychlou konfiguraci v GUI, vizte obrázek.



Nyní již vidíme grafy pro využití paměti RAM a procesorového času.



2.5 Spuštění umělé zátěže v kontejnerizovaném prostředí (společná část)

Závěrem prací připravujících měřené prostředí spustíme a nakonfigurujeme aplikaci zodpovědnou za zátěž. V tomto případě jsem zvolil aplikace Jellyfin, jedná se o jednoduchý mediaserver, který najde využití jak doma, tak i v profesionální sféře - přece jen není nepodobný technologii, kterou využívá Netflix či další streamovací platformy. Protože potřebujeme, aby aplikace přepočítávala rozlišení videa - downscailovala, budou klientská zařízení nastavena tak, aby přijímala obsah v co nejmenším, avšak různém rozlišení. Vzhledem k tomu, že se nejedná o produkční server s veřejnou IP adresou, nemusíme řešit licenční otázky streamovaného obsahu.

Jellyfin spustíme pomocí následujícího docker compose a poté již můžeme změřit potřebné údaje.

```
---
services:
  jellyfin:
    image: jellyfin/jellyfin:latest
    container_name: jellyfin
    ports:
      - "8096:8096"
    volumes:
      - ./config:/config
      - ./jellyfin/cache:/cache
      - /mnt/media:/media:ro
    restart: unless-stopped
```

(How to setup Jellyfin with Docker Compose 2022)

3. Vyhodnocení měření

V této kapitole se zabírám vyhodnocením změřených údajů a jejich komparací, přičemž se zaměřuji na klíčové metriky, které reflektují aktuální stav systému. Jak jsem již v práci uvedl, pro názornost a přehlednost využívám grafy vytvořené pomocí nástroje Grafana. Tyto grafy jsou opatřeny grafickou aparaturou, která umožňuje lepší porozumění některým zajímavým jevům pozorovaným během měření.

Konkrétně se měření zaměřilo na dvě hlavní hodnoty: okamžité absolutní využití procesorového času a faktické množství operační paměti, kterou jednotlivé procesy okupují. Tyto metriky byly sledovány kontinuálně, což poskytlo přehled o dynamice zátěže serveru v reálném čase. Výsledky jsou prezentovány ve formě grafů, jež zobrazují fluktuace využití zdrojů, a tím umožňují identifikovat případné anomálie či náhlé změny v chování systému.

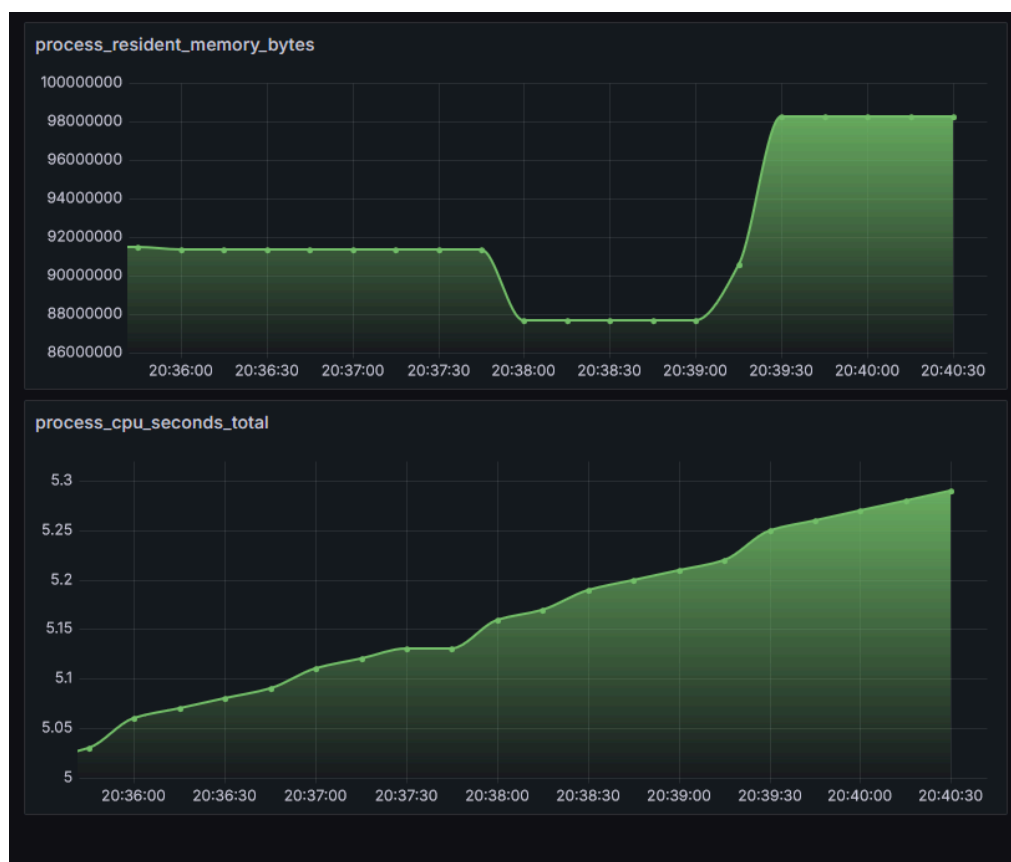
V rámci měření byl rovněž sledován jeden další faktor – kontinuita zobrazení datového toku, respektive obsahu, na klientských stanicích, se kterými server komunikoval. Tento aspekt byl zaměřen na monitorování stabilního přenosu dat mezi serverem a klienty. Nicméně, vzhledem k tomu, že ani já, ani operátoři dalších klientských stanic jsme nezaznamenali jakékoli narušení kontinuity datového toku, není tento údaj považován za směrodatný a nebyl zahrnut do podrobnějšího vyhodnocení.

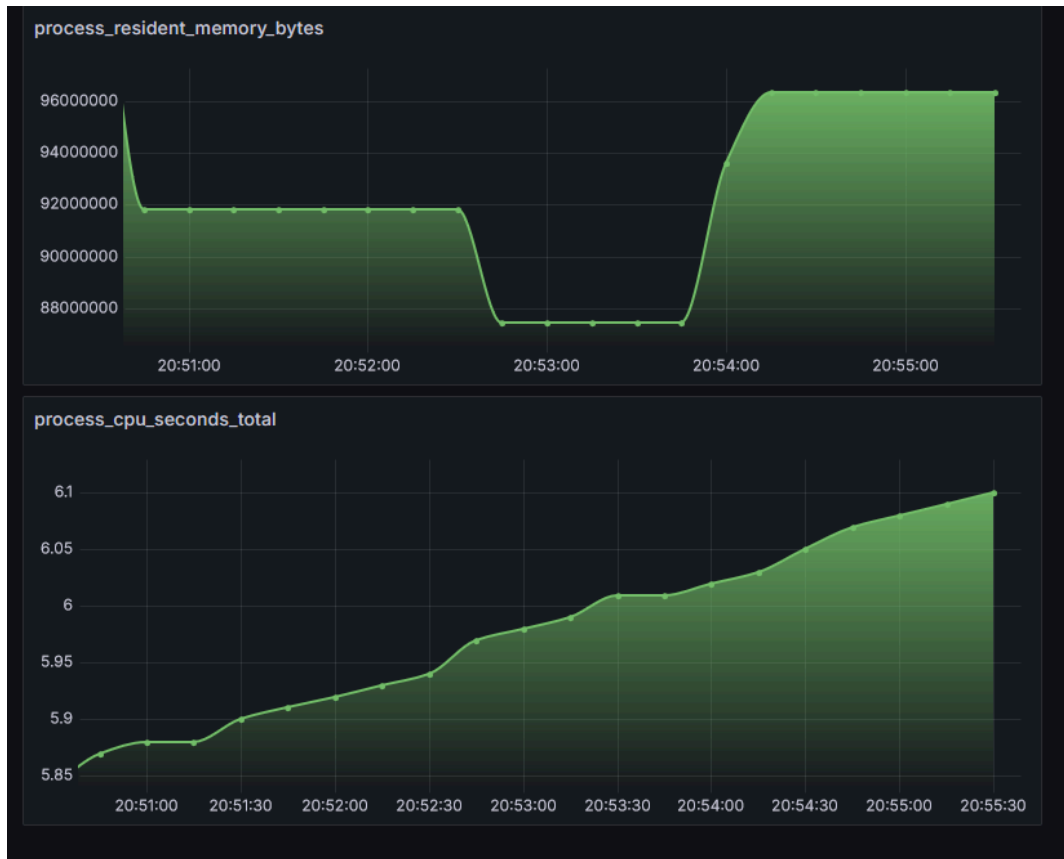
Měření probíhalo v nastavené soustavě, která zahrnovala jeden centrální server, pět klientských stanic umístěných v lokální síti a pět klientů připojených přes internet. Tato kombinace umožnila posoudit výkon a odezvu systému jak v rámci kontrolovaného prostředí, tak i v případě vzdálených připojení, což přináší komplexnější obraz o stabilitě a škálovatelnosti nasazení.

Celkově lze říci, že využití nástroje Grafana výrazně usnadnilo vizualizaci dat a pomohlo identifikovat klíčové trendy a vzorce v dynamice využití systémových zdrojů v komparaci s logy nasazených aplikací. Výsledky měření tak poskytují pevný základ pro další analýzy a optimalizaci provozního prostředí, přičemž potvrzují, že zvolená metodika a použité metriky jsou dostatečně citlivé i pro detekci malých změn v zatížení serveru.

3. 1. Vyhodnocení měření na technologii Docker

Zde jsou umístěné grafy, jejich komentář je následuje dále v práci.







Na výše uvedených grafech je patrné, že procesorový čas narůstá zcela plynule, což lze vysvětlit dvěma hlavními faktory. Za prvé se jedná o poměrně nízké celkové využití procesorového výkonu celé sestavy, což naznačuje, že systém má dostatečné rezervy a nečiní maximální využití všech dostupných jader. Za druhé je patrný i jev, kdy server nedownscaleuje obsah pouze v reálném čase, nýbrž se snaží mírně předbíhat aktuální požadavky. Tento přístup, tedy předběžná alokace zdrojů, je záměrný, protože s rostoucím počtem klientů se výrazně zvyšuje náročnost zpracování dat, a server se tak snaží eliminovat možné zpoždění při jejich doručení.

Teoreticky by bylo možné tento jev dále izolovat či optimalizovat pomocí přesného adresování operací encodingu a scalingových operací na specializovanou grafickou kartu. Takové řešení by totiž umožnilo oddělit úlohy náročné na paralelní výpočty od hlavního procesoru, čímž by se snížila zátěž CPU a zlepšila celková odezva systému. Bohužel, použitá serverová sestava, podobně jako běžně užívané servery, není osazena dedikovanou grafickou kartou, a proto jsou veškeré operace vykonávány výhradně pomocí CPU. Tato skutečnost

může mít vliv na dosažený výkon zejména při zvýšené zátěži, kdy každá mírná odchylka v plánování operací může ovlivnit celkovou efektivitu zpracování.

Přesto je zřetelné, že aktuální konfigurace serveru splňuje požadavky a zvolený přístup je efektivní. Dynamika nárůstu procesorového času zůstává v mezích, které odpovídají očekávanému chování systému při předběžném zpracování dat. Tento kontinuální a plynulý růst zároveň potvrzuje, že systém je schopen pružně reagovat na zvýšený počet klientských požadavků, aniž by došlo k náhlým skokům ve využití zdrojů. Celkově tedy výsledky naznačují, že i bez využití akcelerace pomocí grafické karty je dosaženo vysoké míry optimalizace, přičemž možnost využití GPU akcelerace zůstává perspektivní oblastí pro budoucí zlepšení, zejména v prostředích s extrémně vysokou zátěží.

K zajímavým jevům uvádím následující informace - označené v souladu s grafickým značením grafů:

1. Dokončení přepočítávání aktuálně přehrávaného obsahu pro prvního, v tu chvíli jediného aktivního klienta.
2. Připojení dalších čtyř klientů, v tuto chvíli je aktivních pět klientů, a počátek přepočítávání tří různých obsahů pro pět různých klientů.
3. Server přestal přepočítávat, zde je třeba vnímat, že nedochází k přepočítání celého obsahu, ale jen jeho části pro určitý předstih před reálně odeslanými daty.
4. Zde se aplikace opět pokouší získat předstih před zobrazovaným obsahem
5. Operátor změnil, jaká část obsahu se v tu chvíli zobrazuje - posunul obsah dopředu.
6. Připojení dalších pěti klientů.

3. 2. Vyhodnocení měření na technologii Podman

Zde jsou umístěné grafy, jejich komentář je následuje dále v práci.





Z grafu je patrné, že Podman pracuje s celkovým využitím procesorového času mírně efektivněji než Docker, což naznačuje, že jeho optimalizace běhu procesů je o něco lepší. I když rozdíly nejsou dramatické, lze pozorovat, že Podman se dokáže vyhnout některým výkyvům, které se u Dockeru občas projeví. Obě technologie však už ze své podstaty šetří zdroje poměrně razantní měrou, jelikož princip kontejnerizace vychází z efektivního využití hardwaru.

Dalším významným zjištěním je, že Podman vykazuje podstatně nižší tendenci k využívání operační paměti. Při analýze grafu RAM, zejména během intenzivních provozních událostí, se ukazuje, že hodnoty u Podmanu jsou výrazně stabilnější a méně kolísavé. Docker naopak v těchto obdobích vykazuje větší výkyvy, což může signalizovat vyšší nároky na správu paměťových zdrojů. Tento rozdíl je zvláště pozoruhodný, neboť se předpokládalo, že oba systémy budou mít podobné chování vzhledem k jejich základním principům.

Je tedy zřejmé, že i když jsou rozdíly v celkovém využití procesorového času jen mírné, Podman má jasnou výhodu, pokud jde o konzistentní a úspornější práci s operační pamětí. Menší fluktuace hodnot RAM během náročných momentů naznačují, že Podman lépe řídí alokaci paměti, což se může v praxi projevit vyšší stabilitou celého systému. Tento opak předpokládaného chování, kdy by se očekávalo téměř identické využití zdrojů, poukazuje na to, že drobné rozdíly v implementaci mohou mít v reálném provozu významný dopad na výkon a spolehlivost aplikací.

K zajímavým jevům uvádím následující informace - označené v souladu s grafickým značením grafů:

1. Dokončení přepočítávání aktuálně přehrávaného obsahu pro prvního, v tu chvíli jediného aktivního klienta.
2. Připojení dalších čtyř klientů, v tuto chvíli je aktivních pět klientů, a počátek přepočítávání tří různých obsahů pro pět různých klientů.
3. Server přestal přepočítávat, zde je třeba vnímat, že nedochází k přepočítání celého obsahu, ale jen jeho části pro určitý předstih před reálně odeslanými daty.
4. Připojení dalších pěti klientů.
5. Operátor změnil, jaká část obsahu se v tu chvíli zobrazuje - posunul obsah dopředu.

4. Zpracování výukové prezentace

V rámci práce jsem zpracoval výukovou prezentaci na téma kontejnerizační technologie, která si klade za cíl detailně přiblížit zákoutí této moderní metody nejen z historického hlediska, ale i z pohledu jejího současného využití v produkčním nasazení. Prezentace systematicky rozebírá teoretické základy kontejnerizace – od principů izolace a virtualizace, které umožňují efektivní využití systémových zdrojů, až po konkrétní koncepce a architektury, jež stojí za dnešními technologiemi.

Obsah prezentace začíná historickým přehledem vývoje kontejnerizačních metod, kdy se vysvětluje, jak se myšlenka izolace aplikací vyvíjela od jednoduchých technik až k dnešním sofistikovaným řešením. Dále se věnuji podrobnému rozboru základních komponent, jako jsou kontejnery, imagy a mechanismy jejich orchestrace, což posluchačům ukazuje, jak jsou tyto prvky vzájemně propojeny a jak společně přispívají k optimalizaci IT prostředí.

Důraz je kladen také na strategickou hodnotu kontejnerizace pro moderní IT infrastrukturu. Prezentace ukazuje, jak přístup založený na kontejnerech přispívá ke škálovatelnosti, flexibilitě a rychlosti nasazení aplikací, což je v dnešním dynamickém prostředí neocenitelné.

Celkově prezentace nabízí ucelený přehled od historických souvislostí až po současné praktické aplikace, a poskytuje nástroje a metodiky, které posluchačům umožní lépe pochopit a využít potenciál kontejnerizace v jejich každodenní praxi.

Prezentace i doporučené poznámky řečníka jsou přílohou této práce, prezentace označená jako příloha 1, poznámky k ní jako příloha 2.

5. Zpracování pracovních listů pro výuku kontejnerizace

V rámci práce jsem zpracoval pracovní list, který provází studenty jednoduchým procesem nasazení technologie Docker a spuštěním vybrané aplikace uvnitř této platformy. Tento pracovní list je navržen tak, aby byl co nejvíce srozumitelný a umožnil studentům prakticky si osvojit základní principy kontejnerizace, aniž by se museli potýkat s nadměrnou technickou složitostí.

Pracovní list poskytuje studentům nejen teoretický základ, ale především praktického průvodce krok za krokem, jak nasadit kontejnerizační technologii a spustit v ní vybranou aplikaci. Důraz je kladen na pochopení základních principů a získání praktických dovedností, které jsou klíčové pro moderní správu IT systémů. Studentům se tak otevírá možnost seznámit se s reálným provozem kontejnerů, aniž by museli řešit složitější infrastrukturní záležitosti, jako je například konfigurace reverzního proxy serveru pro směrování domén.

Z důvodu zjednodušení a zaměření se na podstatu problematiky pracovní list záměrně neuvažuje potřebu provozu reverzního proxy. Výstupy z aplikací jsou směrovány přímo na otevřený port, což dále redukuje konfigurační složitost a umožňuje studentům se soustředit na klíčové aspekty práce s Dockerem. Tento přístup zároveň zajišťuje, že nedochází k nežádoucím komplikacím, jež by mohly negativně ovlivnit praktickou výuku ICT.

Pracovní list je přílohou této práce, označený jako příloha 3.

Závěr

Hodnocení měření ukázala, že technologie Docker i Podman jsou schopny efektivně zpracovávat náročné úkoly v reálném čase. Oba nástroje kontejnerizace mají podobné procesorové využití, ačkoli Podman je trochu lepší při využití operační paměti.

V průběhu zkoumání bylo možné pozorovat, jak jednotlivé kontejnerizační nástroje reagují na změnu zátěže při zapojení dalšího klienta, a to většinou bez velkých potíží. Kritické události ukázaly menší odchylky Podmanu ve využití procesorového času a lepší stabilitu spotřeby operační paměti.

Poslední fáze porovnání obou technologií odhalila vyšší stabilitu a efektivitu využití prostředků u Podmanu než u Dockeru, což by mohlo být důležité pro spolehlivost nasazení v průmyslovém prostředí s důrazem na infrastrukturní efektivitu. Nicméně každý nástroj má své specifické funkce a volba mezi těmito dvěma nástroji by měla být provedena na základě konkrétních potřeb projektu. To že rootless Podman vykázal vyšší efektivitu, může být způsobeno podstatně lepšími podmínkami pro vývoj ve velkých společnostech, jako je Red Hat, které Podman vyvíjejí, oproti podmínkám, které má komunitní vývoj Dockeru.

Conclusion

Measurement evaluations have shown that both Docker and Podman technologies are capable of efficiently processing demanding tasks in real time. Both containerization tools have similar processor utilization, although Podman is slightly better at utilizing RAM. During the course of the investigation, it was possible to observe how each containerization tool responds to load changes when another client is involved, mostly without much difficulty. Critical events showed less variation in Podman's CPU time usage and better stability in RAM consumption.

The last phase of the comparison between the two technologies revealed higher stability and resource utilization efficiency for Podman than for Docker, which could be important for reliability of deployments in industrial environments with an emphasis on infrastructure efficiency. However, each tool has its own specific features and the choice between the two tools should be made based on the specific needs of the project. The fact that rootless Podman has shown higher efficiency may be due to the significantly better development conditions in large companies like Red Hat that develop Podman, compared to the conditions that community development of Docker has.

Zdroje

ARORA, Simran. Docker vs. Virtual Machines: Differences You Should Know. Online. In: Cloud Academy. 2023. Dostupné z: <https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/>. [cit. 2023-12-26].

BLAKE, Catherine. 5 Best Practices for Securing SSH. Online. In: Teleport. 2023. Dostupné z: <https://goteleport.com/blog/5-ssh-best-practices/>. [cit. 2023-12-26].

ČECH, Jiří. Kontejnerizace, migrace a škálovatelnost databázových technologií. Online, Bakalářská práce. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2021. Dostupné z: <http://hdl.handle.net/10084/144146>. [cit. 2023-12-23].

DAWSON, Sonikasinghh. Podman vs Docker: All You Need To Know!. Online. In: Dev.to. 2022. Dostupné z: <https://dev.to/lambdatest/podman-vs-docker-all-you-need-to-know-2j9m>. [cit. 2024-05-05].

Debian. Online. 2023. Dostupné z: <https://www.debian.cz/>. [cit. 2024-05-01].

FIGUEIREDO, Rute a GAMELA, Alex. Podman vs Docker: What are the differences? Online. In: Imaginary Cloud. 2023. Dostupné z: <https://www.imaginarycloud.com/blog/podman-vs-docker/>. [cit. 2023-12-24].

Grafana. Online. 2023. Dostupné z: <https://grafana.com/grafana/download?edition=os>. [cit. 2024-05-01].

How to setup Jellyfin with Docker Compose. Online. In: CyberHost. 2022. Dostupné z: <https://cyberhost.uk/how-to-self-host-jellyfin/>. [cit. 2024-05-01].

ŠÍPEK, Michal. Docker kontejnerizace v procesech vývoje SW aplikací. Online, Diplomová práce, vedoucí Ing. Pavel Kříž, Ph.D. Hradec Králové: Univerzita Hradec Králové Fakulta informatiky a managementu Katedra informatiky a kvantitativních metod, 2018. Dostupné z: <https://theses.cz/id/21v8dq/STAG89423.pdf>. [cit. 2023-12-23].

Obrazové zdroje

ARORA, Simran. *Docker vs. Virtual Machines: Differences You Should Know*. Online. In: Cloud Academy. 2023. Dostupné z: <https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/>. [cit. 2023-12-26].